Importing the Dependencies

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from matplotlib import rcParams
import warnings
warnings.filterwarnings('ignore')
```

Data Collection and Processing

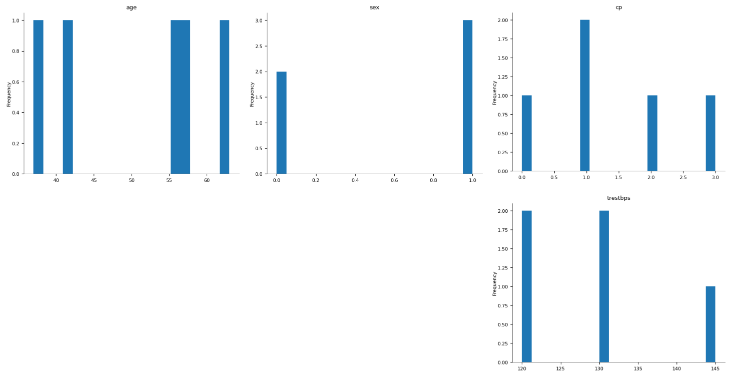
```
#Loading the csv data to a Pandas DataFrame

heart_data =pd.read_csv('/content/data.csv')
```

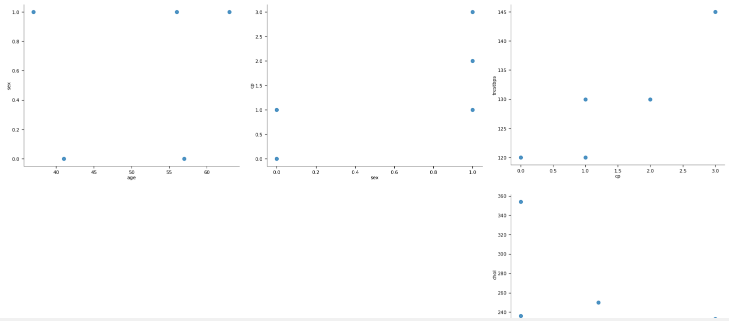
```
#Print first 5 rows of the dataset
heart_data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	tha
0	63	1	3	145	233	1	0	150	0	2.3	0	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	

Distributions



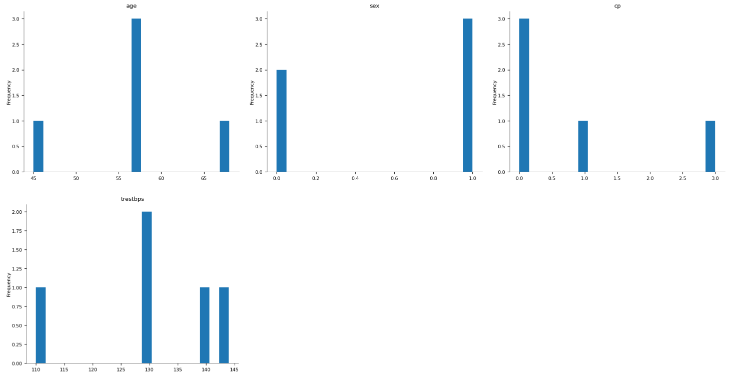
2-d distributions



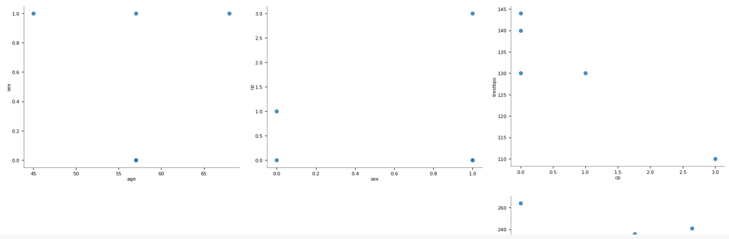
```
#print last 45 rows of the dataset
heart_data.tail()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	t
298	57	0	0	140	241	0	1	123	1	0.2	1	0	
299	45	1	3	110	264	0	1	132	0	1.2	1	0	
300	68	1	0	144	193	1	1	141	0	3.4	1	2	
301	57	1	0	130	131	0	1	115	1	1.2	1	1	
302	57	0	1	130	236	0	0	174	0	0.0	1	1	

Distributions



2-d distributions



```
#The number of rows and columns in the dataset
heart_data.shape
```

```
(303, 14)
```

```
#Getting some info about the data
heart_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
#Checking for missing values
heart_data.isnull().sum()
```

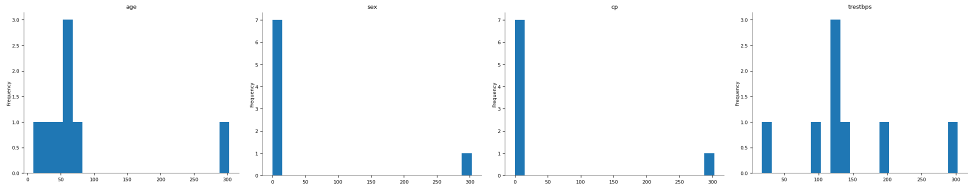
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
```

```
slope       0
ca          0
thal        0
target      0
dtype: int64
```

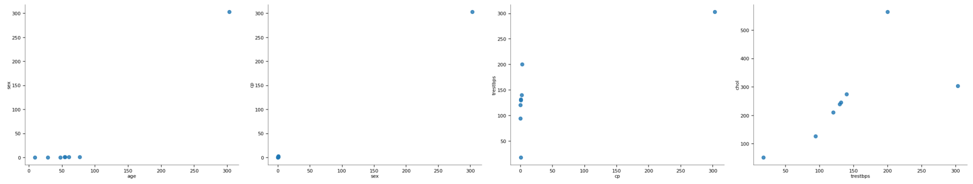
```
#Statsitical measures about the data
heart_data.describe()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.00000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.39934
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.61622
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.00000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.00000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.00000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.00000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.00000

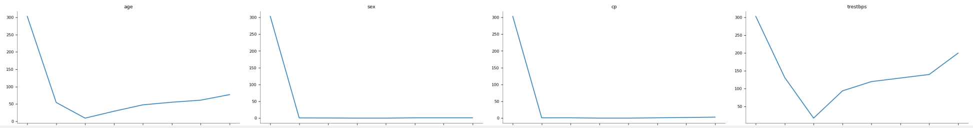
Distributions



2-d distributions



Values



```
#Checking the distribution of target variable
heart_data['target'].value_counts()
```

```
1    165
0    138
Name: target, dtype: int64
```

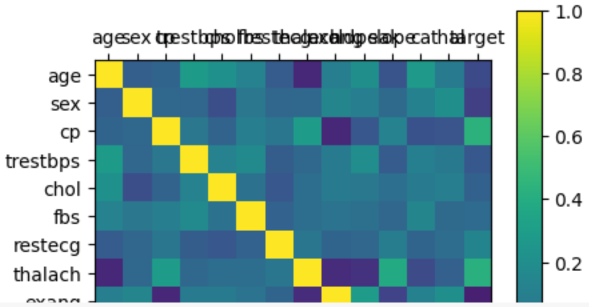
1 --> Defective Heart

0 --> Healthy Heart

Data Visualization

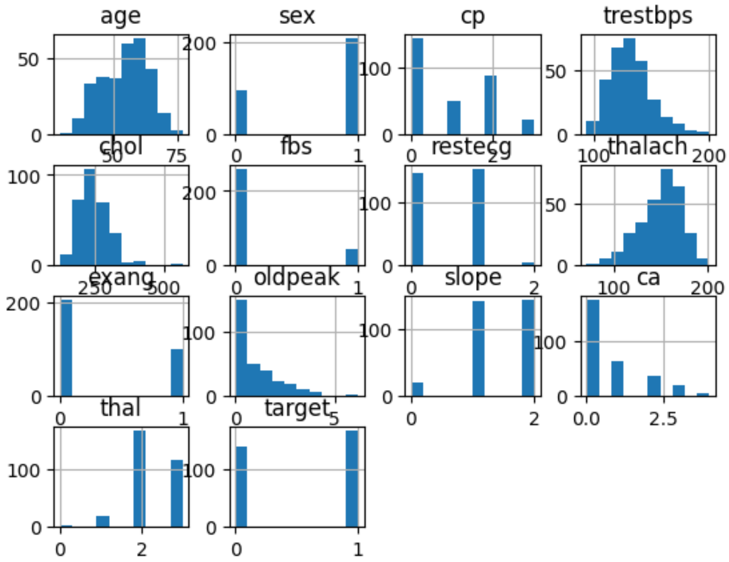
```
plt.matshow(heart_data.corr())
plt.yticks(np.arange(heart_data.shape[1]), heart_data.columns)
plt.xticks(np.arange(heart_data.shape[1]), heart_data.columns)
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7fa7da1fea10>
```



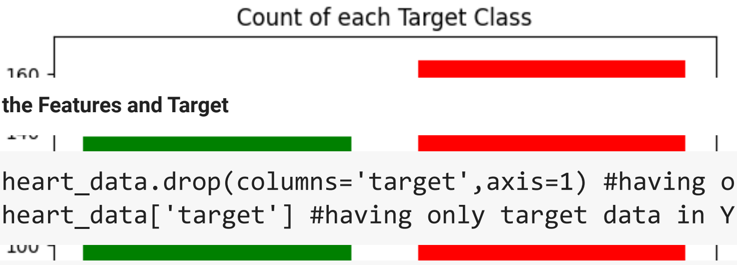
```
heart_data.hist()
```

```
array([[<Axes: title={'center': 'age'}>, <Axes: title={'center': 'sex'}>,
        <Axes: title={'center': 'cp'}>,
        <Axes: title={'center': 'trestbps'}>],
       [<Axes: title={'center': 'chol'}>,
        <Axes: title={'center': 'fbs'}>,
        <Axes: title={'center': 'restecg'}>,
        <Axes: title={'center': 'thalach'}>],
       [<Axes: title={'center': 'exang'}>,
        <Axes: title={'center': 'oldpeak'}>,
        <Axes: title={'center': 'slope'}>,
        <Axes: title={'center': 'ca'}>],
       [<Axes: title={'center': 'thal'}>,
        <Axes: title={'center': 'target'}>, <Axes: >, <Axes: >]],
      dtype=object)
```



```
plt.bar(heart_data['target'].unique(), heart_data['target'].value_counts(), color = ['red',
plt.xticks([0, 1])
plt.xlabel('Target Classes')
plt.ylabel('Count')
plt.title('Count of each Target Class')
```

```
Text(0.5, 1.0, 'Count of each Target Class')
```



Splitting the Features and Target

```
X=heart_data.drop(columns='target',axis=1) #having only feature data in X
Y=heart_data['target'] #having only target data in Y
```

```
print(X)
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1   3       145   233    1        0      150      0      2.3
1     37    1   2       130   250    0        1      187      0      3.5
2     41    0   1       130   204    0        0      172      0      1.4
3     56    1   1       120   236    0        1      178      0      0.8
4     57    0   0       120   354    0        1      163      1      0.6
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
298   57    0   0       140   241    0        1      123      1      0.2
299   45    1   3       110   264    0        1      132      0      1.2
300   68    1   0       144   193    1        1      141      0      3.4
301   57    1   0       130   131    0        1      115      1      1.2
302   57    0   1       130   236    0        0      174      0      0.0

     slope  ca  thal
0        0   0     1
1        0   0     2
2        2   0     2
3        2   0     2
4        2   0     2
..     ...  ..   ...
298      1   0     3
299      1   0     3
300      1   2     3
301      1   1     3
302      1   1     2

[303 rows x 13 columns]
```

```
print(Y)
```

```
0      1
1      1
2      1
3      1
4      1
      ..
298    0
299    0
300    0
301    0
302    0
Name: target, Length: 303, dtype: int64
```

Splitting the Data into Training Data and Test Data

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,stratify=Y,random_state=2)
```

```
print(X.shape,X_train.shape,X_test.shape)
```

```
(303, 13) (242, 13) (61, 13)
```

Model Training

Logistic Regression

```
model=LogisticRegression()
```

```
#Training the LogisticRegression Model with Training Data
model.fit(X_train,Y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

Decision Tree

```
modelD = DecisionTreeClassifier()
```

```
modelD.fit(X_train,Y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

Model Evaluation

Accuracy Score

```
#Accuracy of Logistic Regression Models for Training Data
X_train_prediction=model.predict(X_train)
training_data_accuracy= accuracy_score(X_train_prediction, Y_train)
```

```
#Accuracy of Decision Tree Models for Training Data
X_train_prediction=modelD.predict(X_train)
training_data_accuracyD= accuracy_score(X_train_prediction, Y_train)
```

```
print("Accuracy of Logistic Regression Models for Training Data: ",training_data_accuracy)
print("Accuracy of Decisison Tree Models for Training Data: ",training_data_accuracyD)
```

```
Accuracy of Logistic Regression Models for Training Data:  0.8512396694214877
Accuracy of Decisison Tree Models for Training Data:  1.0
```

```
#Accuracy of Logistic Regression Model for Test Data
X_test_prediction=model.predict(X_test)
test_data_accuracy= accuracy_score(X_test_prediction, Y_test)
```

```
#Accuracy of Decision Tree Model for Test Data
X_test_prediction=modelD.predict(X_test)
test_data_accuracyDT= accuracy_score(X_test_prediction, Y_test)
```

```
print("Accuracy of Logistic Regression for Test Data : ",test_data_accuracy)
print("Accuracy of Decision Tree for Test Data : ",test_data_accuracyDT)
```

```
Accuracy of Logistic Regression for Test Data :  0.819672131147541
Accuracy of Decision Tree for Test Data :  0.7868852459016393
```

Prediction Model

```
input_data=(57,1,0,140,192,0,1,148,0,0.4,1,0,1)

#Change the input  data to a numpy array
input_data_as_numpy_array=np.asarray(input_data)

#Reshape the numpy array as we are predicting for only one instance
input_data_reshape=input_data_as_numpy_array.reshape(1,-1)

prediction=model.predict(input_data_reshape)

if(prediction[0]==0):
  print("The Person does not have Heart Disease")
else:
  print("The Person has Heart Disease")

```

```
The Person has Heart Disease
```

```
import pickle
```

```
filename='trained_model.sav'
pickle.dump(model,open(filename,'wb'))
```

```
#Loading the saved model
loaded_model=pickle.load(open('trained_model.sav','rb'))
```

```
input_data=(57,1,0,140,192,0,1,148,0,0.4,1,0,1)

#Change the input  data to a numpy array
input_data_as_numpy_array=np.asarray(input_data)

#Reshape the numpy array as we are predicting for only one instance
input_data_reshape=input_data_as_numpy_array.reshape(1,-1)

prediction=loaded_model.predict(input_data_reshape)

if(prediction[0]==0):
  print("The Person does not have Heart Disease")
else:
  print("The Person has Heart Disease")

```

```
The Person has Heart Disease
```